Moderne Forms met ADF bij bol.com

Refactoring van de bestaande applicaties

De website van bol.com behoort tot de best bezochte websites van Nederland. De site is in 2011 opnieuw tot beste webwinkel van Nederland gekozen. Met informatie over tien miljoenen producten en een piekbelasting van 120.000 bestellingen op één dag moet niet alleen de website optimaal performen maar ook de backoffice. Deze backoffice heeft sinds de start van bol.com sterke groei meegemaakt, zowel in aantallen als in breedte van het assortiment. Door de aandacht voor de groei was er jarenlang minder aandacht voor zaken als refactoring van de bestaande applicaties.

Aanleiding

De backoffice applicaties zijn ontwikkeld op Oracle databases met een Designer/Forms-technologie voor de GUI. Onder druk van het succes moest er snel ontwikkeld worden. Bol.com bestond uit een handvol medewerkers die alle taken moesten kunnen uitvoeren. Autorisatie, om maar een voorbeeld te noemen, was dus niet direct nodig en een luxe. Inmiddels bestaat bol.com uit meer dan 320 medewerkers die niet meer allemaal alles weten en kunnen. Autorisatie is nu iets dat wordt gemist in deze Oracle Forms schermen. Daarnaast staat bol.com aan de vooravond van een grote omschakeling naar een op SOA gebaseerde architectuur. Bestaande backoffice applicaties zullen daarbij fors onder handen worden genomen.
Deze vragen hebben geleid tot een afweging over de te gebruiken technologie in de backoffice. Vanwege de uitfasering van Oracle Designer en het feit dat Oracle Forms inmiddels toch wel van een zonnige oude dag aan het genieten is, is besloten om over te stappen op ADF.

Projectaanvang

Het ADF-project is gestart met de opleiding van een aantal backoffice-developers tot Java-ontwikkelaars. Naast deze uitgebreide scholing is een specifieke ADF11g-training bij AMIS gevolgd waarna een pilot gestart werd waarin de 'fundamenten' als error-handling etc gestort werden. Deze ontwikkelcapaciteit die in deze ADF-migratie ging zitten ging uiteraard ten koste van de ontwikkelcapaciteit ten behoeve van (nieuwe) business. Om dit te compenseren werd met de business afgesproken dat de nieuwe schermen business proces ondersteunend ontworpen zouden worden. Het voordeel hiervan is dat de gebruikers sneller en efficiënter kunnen werken. Bovendien werd begonnen met nieuwe functionaliteit zodat de toegevoegde waarde van deze nieuwe technologie snel tastbaar zou worden.

De Proof of Concept

Onmiddellijk na de opleiding, en voordat het daadwerkelijke project begon is een proof of concept uitgevoerd. In deze PoC moest een antwoord worden gevonden op de volgende vragen:

1. Is ADF Flexibel genoeg om de back office applicatie in te ontwikkelen ?
2. Kan ADF het gewenste Authenticatie and Autorisatie model aan ?
3. Is het gebruik van JHeadstart een optie ?
4. Kan de JHeadstart forms2 ADF generator ingezet worden ?

De antwoorden op deze vragen waren veelbelovend. De combinatie van JHeadstart en ADF maakte het mogelijk om de gewenste authenticatie en autorisatie eenvoudig te implementeren. De JHeadstart Forms2ADF Generator, een JHeadstart Feature waarmee Forms, tot op zekere hoogte, automatisch in ADF omgezet kunnen worden voldeed niet aan de eisen. De meest complexe Forms module werd niet op de juiste manier door de generator geïnterpreteerd. In principe zou de generator gebruikt kunnen worden voor de eenvoudige forms, maar daar is niet voor gekozen. Juist deze eenvoudige forms zijn een uitstekende leerschool voor beginnende ADF ontwikkelaars. De generator is uiteindelijk alleen gebruikt om voorbeelden te krijgen van wat JHeadstart genereert op basis van de Forms. Deze voorbeelden zijn gebruikt als basis voor de uiteindelijke 'handbouw'.
Het moderniseren van een Oracle Forms applicatie is een

uitdaging op zich. Er zijn legio opties om dit traject aan te vliegen. Geautomatiseerd, met behulp van JHeadstart of eventuele andere tools. Handmatig, door zelf op basis van functionele kennis van de forms applicatie, en technische kennis van ADF de forms modules om te bouwen. Of, misschien wel de beste optie, een (gezonde) combinatie van beiden.
Na deze PoC was het tijd om te beginnen met het daadwerkelijke moderniserings project.

De uitvoering

Het moderniserings project op zich valt uiteen in een aantal hoofdcomponenten.
Ten eerste het opzetten van een robuuste en tevens flexibele applicatie architectuur.

Applicatie Architectuur

In de proof of concept is de benadering gekozen om alle functionaliteit van de applicatie in één grote JDeveloper workspace te bouwen met één Model en één ViewController project. Dit heeft een aantal nadelen.
Op de eerste plaats kan de JDeveloper workspace erg groot worden. Het openen van deze workspace in JDeveloper kan lang duren. Voor beginnende, maar ook voor ervaren ontwikkelaars wordt het lastig om de weg te vinden in deze structuur. Het bij elkaar zoeken van technische delen die bij één functionele eenheid horen wordt hierdoor moeilijk. Daarnaast loop je het risico dat meerdere ontwikkelaars dezelfde bestanden gaan wijzigen waardoor conflicten ontstaan.

Na de eerste fase is besloten om de applicatie architectuur te herzien. De grote workspace is gerefactored naar meerdere kleine en modulaire workspaces, en de nieuwe te bouwen ADF functionaliteit is ook volgens deze aanpak ontwikkeld. Het werken met kleine functionele eenheden (lees: 1 forms module wordt 1 applicatie) heeft een aantal voordelen.

Best Practice

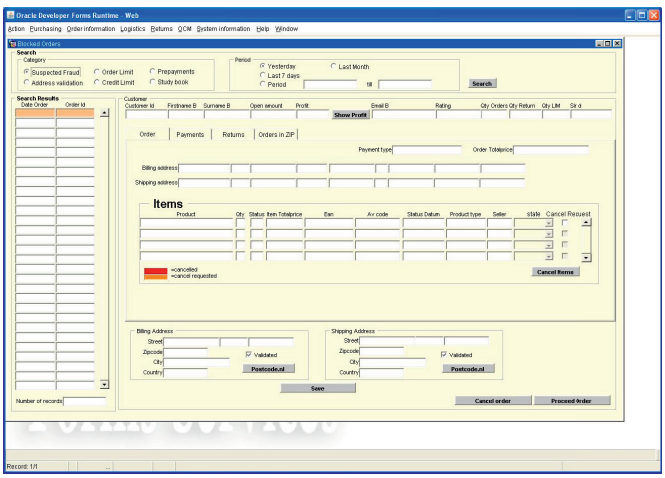
Deze aanpak wordt in de ADF community gezien als 'good' zo niet 'best practice'. Ten eerste laden ze sneller in JDeveloper. Daarnaast zijn ze overzichtelijk. Ze bevatten immers één functionele eenheid, dus alles hoort bij elkaar. Verder kunnen meerdere ontwikkelaars parallel werken aan de applicatie zonder in elkaars vaarwater te komen.
Het belangrijkste voordeel van het werken op deze manier is de mogelijkheid tot hergebruik. De kleine applicaties worden gedeployed als ADF taskFlow Library en kunnen, daar waar nodig, worden hergebruikt. De ADF Taskflow libraries komen samen in één 'centrale' applicatie. Deze consumeert de taskflows in een dynamische regio die worden aangestuurd vanuit een menu.

Figuur 1: menu

Alleen díe taskflow (lees: gebruikers functie) die door de gebruiker in het menu wordt gekozen zal door de applicatie worden geladen en getoond.

User Interface en lay-out

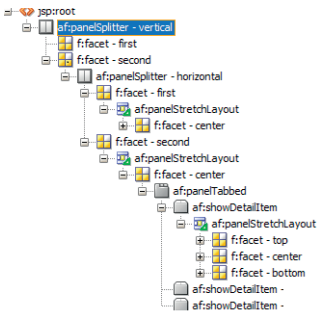
Hoe moet de applicatie er uit gaan zien, en hoe kan dat resultaat het beste bereikt worden ? Nadenken over de User Interface houdt niet op bij de look and feel. Voor alle forms componenten moet een ADF alternatief worden gevonden of gemaakt. Het maken van de lay-out voor een ADF applicatie is lastig. Pixel perfect ombouwen van een Forms applicatie naar ADF kan daardoor zeer bewerkelijk, en soms zelfs onmogelijk zijn. Daarom is geprobeerd om binnen de grenzen van het reële, de 'oude' forms applicatie te benaderen qua lay-out, maar deze zeker niet ná te bouwen.
Om dit te doen kan de bestaande applicatie ontleedt worden in functionele visuele componenten. Als voorbeeld één van de complexe forms uit de applicatie.



Figuur 2: Originele Form

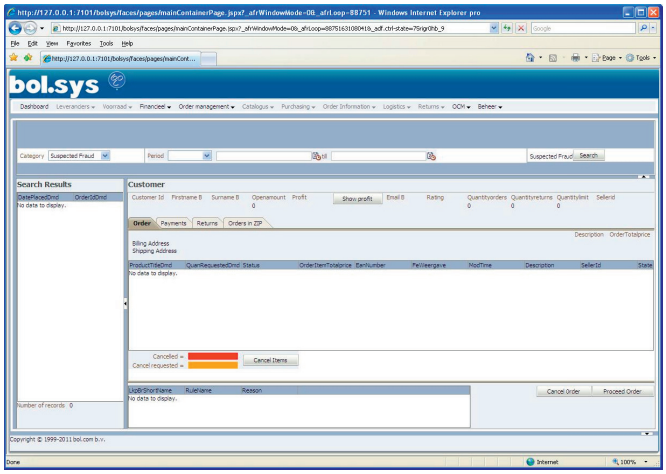
Deze form bevat een aantal duidelijk zichtbare onderdelen: Een zoekblok, een Master Resultaat Block Table, een Master Resultaat Block Form, een Tabbed Panel, een Detail Resultaat Block (Form and Table), een Extra Detail Resultaat Block, items die totalen tonen en tenslotte een menu om de applicatie aan te sturen.
Al deze onderdelen moeten terug komen in de ADF Applicatie. Het probleem in deze specifieke forms module zit niet in het tonen van Master-Details met de bijbehorende data. Deze functionaliteit biedt ADF 'out of the box'. Met behulp van ADF

Business Components wordt de data uit de database ontsloten, en de implementatie van Master Details is recht toe recht aan.
De complexiteit ligt vooral in het bouwen van de layout, en het gebruik van de ADF Layout containers. Daarnaast is de implementatie van de zoekfunctionaliteit en de PL/SQL achter de verschillende knoppen lastig. Dit komt later in dit artikel aan de orde.
Voor het bouwen van de layout is kennis van ADF Layout containers erg belangrijk.



Figuur 3: structuur

Door gebruik te maken van de PanelSplitter and de PanelStretchLayout componenten werd de layout van de forms module goed benaderd. Daarnaast is de panelTabbed component ingezet. Deze is een perfect alternatief voor de vele tabbed canvassen die in de Oracle Forms Applicatie aanwezig waren.



Figuur 4: ADF Scherm

Implementatie van de PL/SQL code

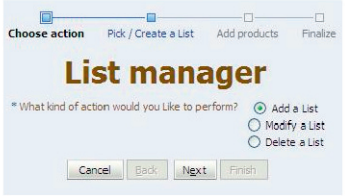
De User Interace is pas het begin. Het zou toch mooi zijn als de ADF applicatie dezelfde functionaliteit heeft als de oorspronkelijke Forms Applicatie. Daarbij speelt de vraag "hoe implementeer je forms (PL/SQL) logica in een ADF (= J2EE) applicatie" een grote rol. PL/SQL code is geen Java. Gelukkig is dat voor niemand een verrassing. Deze code is dan ook met geen mogelijkheid te migreren naar Java. Er zijn bedrijven die pretenderen dat dit wel kan, maar de kwaliteit en onderhoudbaarheid van de Java code die dat oplevert is ver beneden de maat. Toch zal de PL/SQL code op een of andere manier moeten worden overgenomen in de ADF Applicatie.
Voor alle PL/SQL code moet een aantal opties worden bekeken. Op de eerste plaats is uitgezocht of het mogelijk was om de PL/SQL code te verplaatsen naar de database. Als dat het geval is dan is dit veruit de beste en makkelijkste optie. De code kan bijna 1 op 1 worden overgenomen en worden opgenomen in een (al dan niet packaged) stored procedure of stored function. Deze stored procedure of stored function kunnen zonder problemen worden aangeroepen vanuit een ADF applicatie.

```
public void proceedOrder(String p_order_id, String p_old_status, String
p_new_status) {
        String statement = "ocm_blocked_orders_module.proceed_
order(?,?,?)";
        DbUtils.callStoredProcedure(statement,

new Object[]{p_order_id, p_old_status, p_new_status},
                                        getDBTransaction(), true);
}
```

Er zijn situatie waarin je tegen beperkingen van de jdbc driver aan loopt, maar deze zijn zonder al te veel problemen op te lossen. Een voorbeeld hiervan is dat de jdbc driver geen gebruik kan maken van een stored function die een boolean als uitkomst heeft. De workaround ligt voor de hand.
De tweede optie is het zoeken naar 'zero code' alternatieven. De implementatie van Wizard Functionaliteit in forms gaat gepaard met veel PL/SQL code. ADF biedt door middel van zogeheten 'train Task Flows' een alternatief waar nauwelijks (soms zelfs helemaal geen) code aan te pas komt.

Forms Wizard



ADF Wizard

Figuur 5 : wizard

Een ander voorbeeld is zoeken in ADF applicaties. ADF biedt drie (of eigenlijk 4) manieren waarop je zoekfunctionaliteit kunt implementeren.

1) De ADF Query (and Quick Query) componenten.
2) De ADF Search Form
3) De ADF Parameter Form

Al deze alternatieven zijn gebruikt in het project. Afhankelijk van de functionele eisen gebruik je een andere implementatie. De ADF Query component is een 'rijke' component en biedt veel 'out of the box' functionaliteit. De implementatie is eenvoudig en snel (drag en drop). De flexibiliteit van deze component is daarentegen vrij beperkt. Als er afgeweken moet worden van de standaard implementatie wordt het al vrij snel erg lastig.
Het ADF Search Form is niet zo 'rijk', maar kan snel en makkelijk (drag en drop) gebouwd worden en biedt functionaliteit zoals 'enter query mode' en 'execute query mode' waarmee de forms gebruikers bekend zijn..
Bovenstaande zijn allemaal min of meer zero code ADF alternatieven voor zoek functionaliteit. In de gevallen waarin de twee Database implementaties en de Zero Code alternatieven niet gebruikt kunnen worden moet een alternatief in Java code worden bedacht. Deze situatie wordt beschreven aan de hand van 'zoek' functionaliteit zoals deze in één van de forms aanwezig was.
De ADF parameter form is vanuit het oogpunt van implementatie moeilijker dan de eerder genoemden. Daarentegen levert deze component verreweg de meeste flexibiliteit

Figuur 6 : zoekblok

De functionaliteit achter de zoekknop wordt geïmplementeerd door een Java methode die alle zoekcriteria als parameter binnenkrijgt. In deze methode wordt de uit te voeren query samengesteld. Deels door het toepassen van view criteria (1)

```
if(pStockAvailable.equalsIgnoreCase("S")){
    vo.applyViewCriteria(getViewCriteria("stockAvailable_S_vc"),true);
  }
```

Deels door het zetten van bind variabelen (2)

```
if(pListId.equalsIgnoreCase("WORK_ID")){
          ViewCriteria workIdVc = getViewCriteria("workId_vc");
          workIdVc.resetCriteria();
          VariableValueManager workIdVvm = workIdVc.ensureVariable
Manager();
          workIdVvm.setVariableValue("b_workId", pIdentifier);
vo.applyViewCriteria(workIdVc,true);
        }
```

en zelfs door het toevoegen van custom where clauses (3) aan de query.

```
if(pOption.equalsIgnoreCase(QueryUtils.getCn_top_sales_short())){
      String pTopSalesMax = QueryUtils.getTopSalesMax(trans);
      addFragmentToWhereClause(" pos <= to_number(nvl("+pTop +","+
pTopSalesMax +" ))and  ");
  }
```

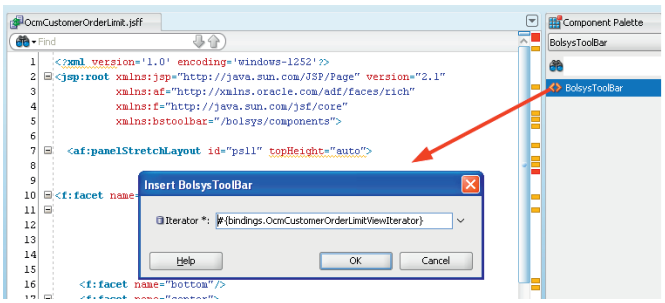
Door deze benadering is het veel eenvoudiger om op runtime nivo queries samen te stellen en uit te voeren op basis van door de gebruiker opgegeven voorwaarden. De zoekfunctionaliteit uit de oorspronkelijke form kon op deze manier exact worden gereproduceerd.

Herbruikbare Componenten

Binnen een groot project is de kans groot dat er componenten zijn die in meer dan één pagina moeten worden gebruikt. Er zijn dan twee opties. Als eerste kun je deze component meerdere malen bouwen. Dit houdt wel in dat je bij wijzigingen ook meerdere implementaties moet aanpassen. De tweede optie is het maken van een Custom Declarative Component (CDC). In dit Forms Moderniserings traject zijn meerdere CDC's ontwikkeld, maar de custom toolbar is veruit de belangrijkste en meest complexe. Deze toolbar bevat een behoorlijk aantal regels complexe code. Het voordeel is echter dat deze toolbar, nadat hij is gedeployed als ADF library, op een eenvoudige manier ingezet kan worden.
Deze toolbar is gemaakt om de applicatie te voorzien van een 'Oracle Forms'-achtige toolbar en komt terug op alle pages die een tabel bevatten.

Figuur 7 : toolbar

De toolbar bevat naast een blader mechanisme ook functionaliteit om records toe te voegen en te verwijderen. Daarnaast



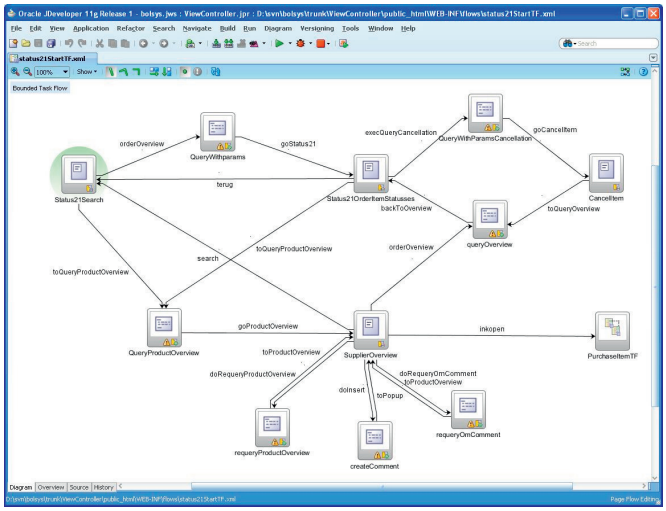
Figuur 8 : use toolbar

uiteraard commit en rollback knoppen. Voor alle componenten op deze toolbar kan worden aangegeven of deze al dan niet moeten worden getoond.
Bij het opnemen van de toolbar in een pagina hoeft alleen maar te worden aangegeven voor welke datacollectie de functionaliteit bedoeld is.

En verder....

Nieuwe Functionaliteit

Naast de modernisering van de Forms applicatie is er nog veel meer gebeurd. Zo is er bijvoorbeeld nieuwe functionaliteit ontwikkeld. Naast eenvoudige invoer modules is ook nieuwe functionaliteit gebouwd die complexe backoffice processen implementeert. Met behulp van ADF taskflows was het mogelijk om deze flow door deze processen dusdanig te bouwen dat eindgebruikers op een voor hun logische manier door de schermen van de applicatie worden geleid in plaats van zelf op zoek te moeten gaan naar de volgende stap in het proces.



Figuur 9 : taskflow

Styling

Aan het stylen van de applicatie is niet heel veel tijd besteed. De meeste tijd is gaan zitten in het bouwen van de daadwerkelijke functionaliteit, en over de look en feel maakte niemand zich druk. Dat veranderde echter direct nadat de eerste eindgebruikers de applicatie voor het eerst zagen. Met minimale effort (4 daagse time box) is het toch gelukt om een BOL. COM-achtige look and feel neer te zetten. Een 'page template' zorgt voor een consistente look and feel door de hele applicatie en met behulp van skinning en een beetje CSS is dit goed gelukt. Meest in het oog springen de BOL- blauwe kleur en de BOL logo's op diverse plaatsen in de applicatie.

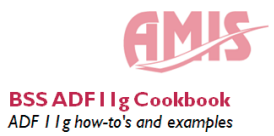
Figuur 10 : styling

Ontwikkelstraat

Het inrichten van de ontwikkelstraat is een verhaal apart. Het gaat te ver om dit in dit artikel helemaal te beschrijven. Versiebeheer (Subversion), Unittesting (JUnit) en Continuous Integration (Hudson / Maven) moeten in principe aan het begin van een project op orde zijn. In dit project is unittesting en versiebeheer direct ingezet, maar door het ontbreken van een buildserver is Continuous Integration pas in een later stadium ingericht.

Ontwikkelstandaarden en Richtlijnen (CookBook)

Naast een goede architectuur zijn standaarden en richtlijnen onmisbaar bij het bouwen van een ADF applicatie. ADF biedt zoveel mogelijkheden dat het belangrijk is dat goed gedocumenteerd wordt hoe bepaalde functionaliteit wordt geïmplementeerd. Waar staan de knoppen op een scherm ? Welke alternatieven voor welke Forms functies ? Hoeveel records worden getoond in een tabel ? Wat is de layout van een tabel ? Hoeveel records kunnen worden geselecteerd in een tabel ? Heeft een tabel filters ? Kunnen records in een tabel worden gewijzigd ?



Figuur 11 : cookbook

Dit lijken niet alleen simpele vragen, dat zijn het ook. Maar als je geen standaarden definieert dan wordt je applicatie al snel een moeras voor de eindgebruiker omdat iedere ontwikkelaar alles op zijn eigen manier implementeert. Tijdens het project zijn heel veel how to's, do's en dont's, best practices en richtlijnen beschreven en opgenomen in een ADF Cookbook. Dit kan worden gebruikt in toekomstige projecten, maar ook als referentie en technische documentatie.

Development Lessons Learned

Durf de gekozen architectuur te herzien.

Architectuur is de basis voor een robuuste en flexibele applicatie. Na de proof of concept bleek dat de architectuur van de applicatie niet flexibel was. We hadden op de ingeslagen weg door kunnen gaan, maar dat is niet gebeurd. De architectuur is volledig gereviseerd en vanaf dat moment is alles in de nieuwe architectuur ontwikkeld. Dit was een risico, maar deze switch heeft een positief effect gehad.

Probeer niet om de Forms Applicatie één op één na te bouwen.
Het één op één nabouwen van een Oracle Forms Applicatie is vreemd. Het is niet alleen erg lastig, maar het dient geen enkel doel. Ten eerste moet er heel veel werk gestoken worden om functionaliteit te bouwen die heel makkelijk te maken is in Forms maar heel lastig is in ADF. Probeer vooral de discussie aan te gaan om lay-out wijzigingen en eventuele minimale functionele wijzigingen door te voeren. Dit kan een heleboel tijd besparen. De eerder beschreven zoekfunctionaliteit is hier een goed voorbeeld van. Ten tweede kun je bij 1 op 1 herbouw geen gebruik maken van de web 2.0 componenten uit ADF 11g. Als je de ADF componenten goed kent kun je juist flexibele en dynamische applicaties bouwen met typische web 2.0 functionaliteit.

Praat met eindgebruikers voor dat je begint.
Eindgebruikers hebben een eigen manier van werken met de applicatie. Ze doen vaak andere dingen dan verwacht en gebruiken sommige delen van de applicatie helemaal niet. Een praktijkvoorbeeld: Het zoeken van een boek op basis van een deel van de titel duurde in de Forms Applicatie oneindig lang. Wat bleek: De gebruikers weten dat en deze functionaliteit werd nooit gebruikt. Wat doen ze dan wel: Surf naar bol.com, zoek op de website (supersnel), kopieer het ISBN nummer en zoek in backoffice applicatie op (Primaire Sleutel) ISBN.
Met deze kennis was de investering in het bouwen van de zoekfunctionaliteit op (deel van) de titel niet nodig geweest.

Automatische Migratie is niet eenvoudig.
Automatische migratie voor de implementatie van eenvoudige tabellen of forms kan eenvoudig zijn, maar voor de meer complexe onderdelen van de Forms applicatie zul je op zoek moeten naar ' the best of both breeds '. Voor een deel handmatige herbouw, en voor een deel geautomatiseerd. In alle gevallen zul je situatie tegenkomen waarin je migratie tool er niet meer uitkomt. En hoe je het ook wendt of keert, de PL/SQL code zal altijd opnieuw moeten worden geschreven in Java.

Lessons learned

Vanuit projectmatig oogpunt is de keuze voor ADF11g een goede keuze geweest. Als framework is het zeer productief met een rijke user interface. Het werkt goed samen met de database; een belangrijk aspect aangezien veel van de business logica van bol.com in de database vastgelegd is. Gebruikers zijn tevreden, direct database access door (een deel van de) gebruikers is verleden tijd. De technologie is SOA-ready waarmee de investeringen geborgd zijn. Niet onbelangrijk, tenslotte, is het feit dat ADF supported technologie betreft waarmee ondersteuning op op langere termijn verzekerd is.
Dit wil niet zeggen dat er geen leerpunten zijn geweest.
Een duidelijk leerpunt was de zeer steile leercurve voor de ontwikkelaars. Door de lengte van deze tweede van de vier leerfases (van onbewust-onbekwaam via bewust-onbekwaam naar bewust-bekwaam naar tenslotte onbewust-bekwaam) lag het gevaar van een motivatie-dip op de loer. Dit werd ondervangen door extra inhuur van ADF-coaches die de medewerkers in hun leertraject konden begeleiden. Dit heeft goed gewerkt. Een onbedoeld neveneffect was dat de momenten waarop de coach niet actief aan het coachen was hij zich bezig hield met de migratie van Forms-schermen naar ADF-technologie. Hierdoor is in korte tijd een groot deel van de te migreren schermen overgezet naar ADF-technologie.
Een ander duidelijk leerpunt voor ons was het belang om de business snel van tastbare resultaten te voorzien. Bij bol.com had met name het inrichten van de complete OTAP-straat, inclusief synchronisatie van de Ms Active Directory met de Oracle Internet Directory, behoorlijk wat voeten in de aarde. Niet in de laatste plaats doordat een deel van de soft- en hardware bij een hosting provider draait en een ander deel bij bol.com op het kantoor in Utrecht.
Door de tijdsduur tussen de start van de pilot en de in productiename zat ruim een jaar. In de wereld van de e-commerce is dit een enorm lange tijd. Toch is deze periode nog aan de snelle kant wanneer rekening gehouden wordt met het feit dat de ADF-ontwikkelaars naast hun ADF-werk ook het reguliere ontwikkelwerk hadden.
Terugkijkend kunnen wij concluderen dat de overgang naar ADF 11g hobbelige weg naar succes is geweest.

Anthonie van Dijk is teamleider Oracle Development bij bol.com. Bol.com is gevestigd in Utrecht. Inmiddels heeft deze e-retailer meer dan 3 miljoen actieve klanten die in 2010 meer dan 16 miljoen artikelen gekocht hebben. De IT-afdeling van bol.com bestaat uit meer dan 100 medewerkers waarvan een groot deel in Development actief is. Binnen bol.com wordt volgens Scrum gewerkt.

Luc Bors (luc.bors@amis.nl) werkt als technisch specialist/architect en is ADF Expertise Lead bij AMIS Services. Hij ontwikkelt ADF workshops en trainingen en is ADF en JHeadstart trainer. Luc presenteert op (inter-) nationale conferenties en publiceert regelmatig artikelen over ADF gerelateerde onderwerpen in diverse tijdschriften en op de AMIS technology blog (http://technology.amis.nl/blog) en op OTN (http://www.oracle.com/technetwork/articles/bors-adfmobile-086867.html).